REMARKS

In response to the Office Action mailed on July 16, 2008, Applicant(s) respectfully request(s) reconsideration.

Claims 1-9, 12-17, 20-28, 30, 31 and 33-49 are now pending in this Application.

In this Amendment, claims 1, 4, 33, 34, 38, 49 and 51 been amended and claim 37 has been cancelled and claims 52-54 have been added.

Claims 1, 33, 34, 49 and 51 are independent claims and the remaining claims are dependent claims. Applicant(s) believe that the claim(s) as presented are in condition for allowance. A notice to this affect is respectfully requested.

Formalities:

The Office Action rejects claims 1-9, 12-17, 20-28, 30, 31 and 33-49 under **35 U.S.C. §112**. Clarification and support for the claimed underlying delivery infrastructure may be found in the specification as filed at page 7, lines 19-27 and at page 10, lines 23-26. Accordingly, it is respectfully requested that the rejection under **35 U.S.C. §112** be withdrawn.

Claim 49 has been herein amended to clarify a computer readable medium configuration.

Rejections under 35 U.S.C. §103:

The Office Action rejects Claims 1-9, 12-17, 20-28, 30, 31, 33-49 and 51 under **35 U.S.C. §103** as being obvious over Silbershatz, et al. (2000), (hereinafter Silbershatz) in view of Frank, U.S. Pub. No. 2004/0250254 (hereinafter Frank '254). For the reasons discussed below, neither Silbershatz nor Frank '254 discloses, alone or in combination, the invention as now expressed by the amended claims. Accordingly, it is respectfully requested that the rejection under 35 U.S.C. 103 be withdrawn in view of these remarks and amendments.

One of the objects of the claimed system is to provide selective invocation, or loading, of components in response to invocation of currently inactive components. In particular, in object oriented development environments such as a conventional CORBA (Common Object Requires Broker Architecture) environment, invocation of inactive (not currently loaded in memory) components (objects) requires identification of and loading of the invoked component. Development of such identification and loading tends to consume developer time; accordingly, the presently claimed activation mechanism provides selective loading of the inactive components, thereby relieving developers of this burden. Amended features of the claims, discussed below, further emphasize this distinctive activation mechanism, disclosed at page 5, lines 21-25. The activation mechanism, as now presently claimed, selectively loads modules from an inactive, nonexecutable location such as a nonvolatile disk file to an executable memory, in contrast to the thread enabling context switch of Frank '254.

Specifically, with respect to claim 1, Silbershatz is cited for the proposition of teaching, inter alia, activation and deactivation operations as recited in claim 1. Silbershatz, however, merely provides a survey of general operating system concepts. Silbershatz does not show, teach, or disclose selective loading in response to object invocation, but rather loading in response to OS scheduling algorithms. The disclosed Silbershatz concepts of swapping, loading and overlays refer to <u>OS initiated</u> action, not <u>application initiated</u> actions such as component invocation in a CORBA environment that requires the selective loading of invoked components as disclosed and claimed herein, discussed further at page 4, line 30-page 5 line 13.

Frank '254 refers to thread invocation, not component invocation. Frank does not perform loading of components; Frank performs activation of threads, as clarified at paragraphs [0065-0068]. In contrast, in the claimed system, loading is not performed at a thread level of granularity; rather, loading is performed on a component which may include many threads. While Frank '254 discloses a thread context, such a thread is already loaded, meaning that the code containing the thread has been transferred from disk into volatile memory, or RAM, as is known in the art. Frank does not show, teach, or disclose selective loading of a component (or module, the coded instructions representing the component). Accordingly, one of skill in the art would not look to Frank '254 to modify the Silbershatz process activation because the Silbershatz activation refers to page swapping to non-volatile (disk) storage (sec. 20.6, p. 642-646) while Frank teaches context switching of threads already loaded in memory, discussed at paragraphs [0080-0086]. Further, even if one of skill were to attempt such a combination, the claimed invention would still not be realized because the memory footprint conservation provided by the claimed activation mechanism would be inoperable with the context switching of Frank '254.

In contrast to the claimed <u>component </u>activation mechanism, Frank teaches thread operations in thread processing units (TPU) 10-20 disposed between a memory cache 26 and processing units 30-38 [0057]. Fig. 3, depicting the alleged anticipative loading operation of paragraph [0082], indicates that threads transition between states of active and inactive, but are nonetheless resident ("loaded") in the TPU whether executing instruction or not [0080-0086]. Memory operations on the TPUs are performed via cache accesses [0125]. Thread execution of active threads is performed via a cache instruction fetch [0187]. Thus, thread state transition of Fig. 3 does not correspond to loading of individual threads, but rather memory operations (context switches) of loaded threads. In Frank '254, since the TPUs 10-20 are between the caches 22,24 and processor units 30-38, threads therewithin have already been loaded; no selective loading as described and claimed in the disclosed system is performed

Therefore, neither Frank '254 nor Silbershatz teaches, alone or in combination, <u>selectively enabling a module, if the module including the corresponding subscriber and event handler is disabled</u>, as now recited in amended claim 1, formerly recited in claim 4 and disclosed in the specification at page 9, line 30-page 10 line 6. Further claim 52 had been added to clarify that activation and deactivation corresponds to loading and unloading (unlinking) a component, as clarified at page 20, lines 24-29. Further, claim 53 makes clear that the modules are <u>code instructions that define the component</u>, and that <u>each</u>

component includes a plurality of threads, to further distinguish the thread based state changes of Frank, discussed at page 12, lines 11-31. Still further, claim 54 has been added to clarify that the activation and deactivation is initiated by application action, not OS action as taught by the Silbershatz reference, as discussed at page 7, lines 16-29.

The remaining independent claims 33, 34, 49 and 51 have been amended with the features as per claim 1 above, and are likewise therefore also believed allowable for the reasons given above. As the remaining claims depend, either directly or indirectly, from claims 1 and 34, it is respectfully submitted that all claims are now allowable.

Applicant(s) hereby petition(s) for any extension of time which is required to maintain the pendency of this case. If there is a fee occasioned by this response, including an extension fee, that is not covered by an online payment made herewith, please charge any deficiency to Deposit Account No. 50-3735.

If the enclosed papers or fees are considered incomplete, the Patent Office is respectfully requested to contact the undersigned collect at (508) 616-9660, in Westborough, Massachusetts.

Respectfully submitted,

/CJL/

Christopher J. Lutz, Esq.
Attorney for Applicant(s)
Registration No.: 44,883
Chapin Intellectual Property Law, LLC
Westborough Office Park
1700 West Park Drive
Westborough, Massachusetts 01581
Telephone: (508) 616-9660
Facsimile: (508) 616-9661

Attorney Docket No.: EMC03-22(03111)
Dated: October 16, 2008